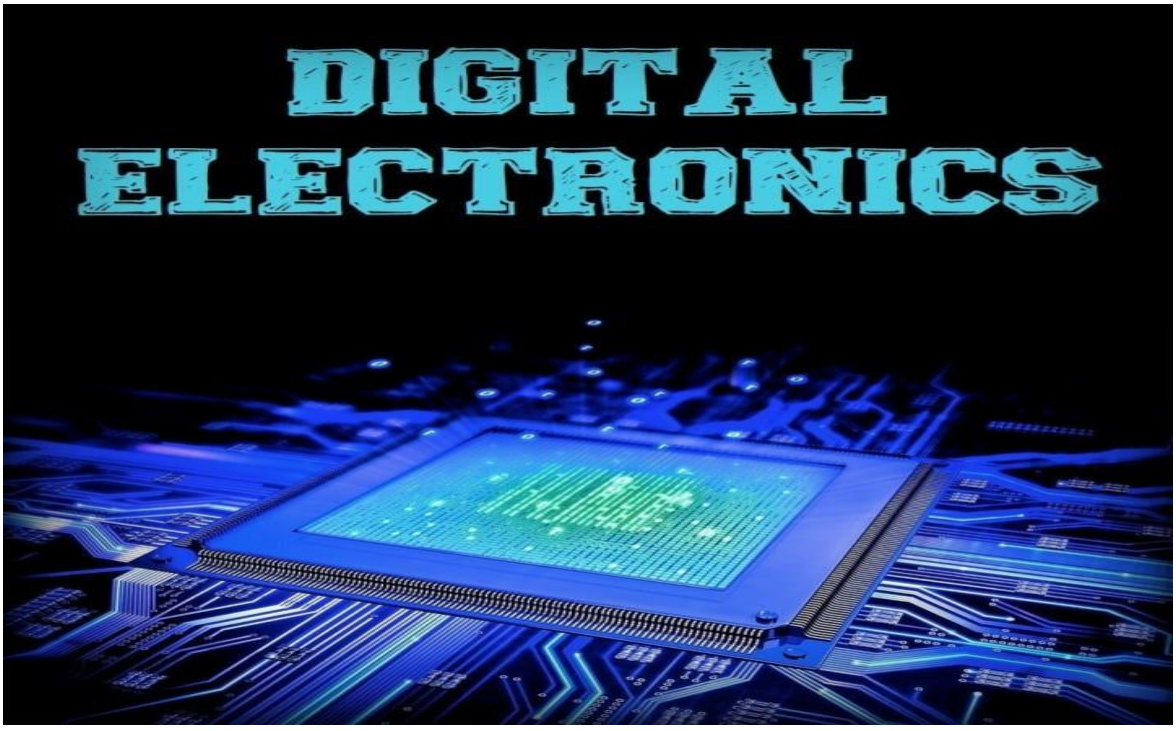
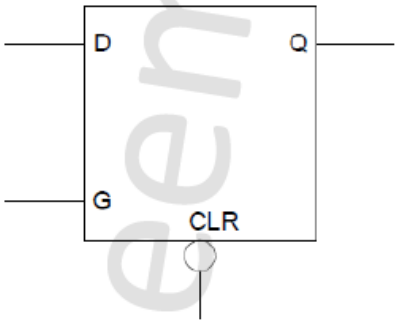
Abanob Evram

Assignmen3



1) Implement Data Latch with active low Clear



[Q1]

The design code:

```
module Data_Latch(CLR,D,G,Q);
input CLR,D,G;
output reg Q;
always @(*) begin
if (~CLR)
Q<=0;
else if (G)
Q<=D;
end
endmodule
```

Input	Output
CLR, D, G	Q

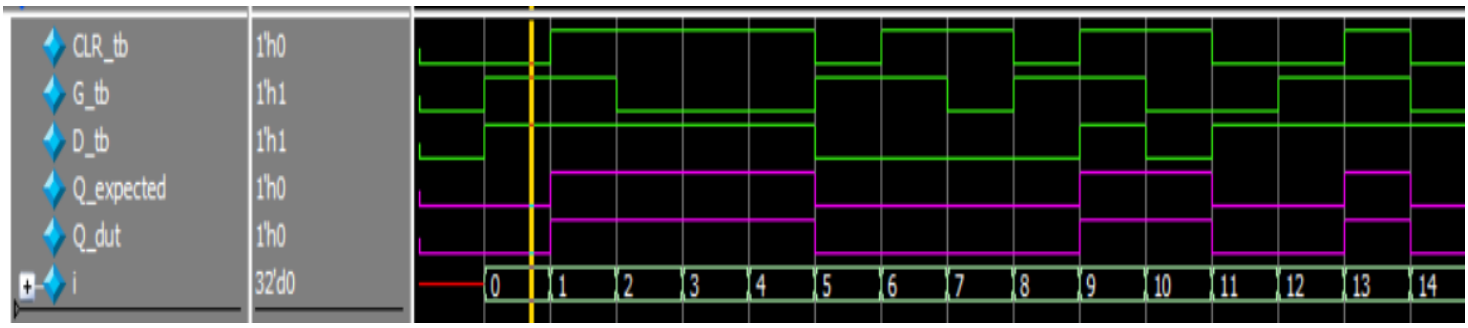
Truth Table

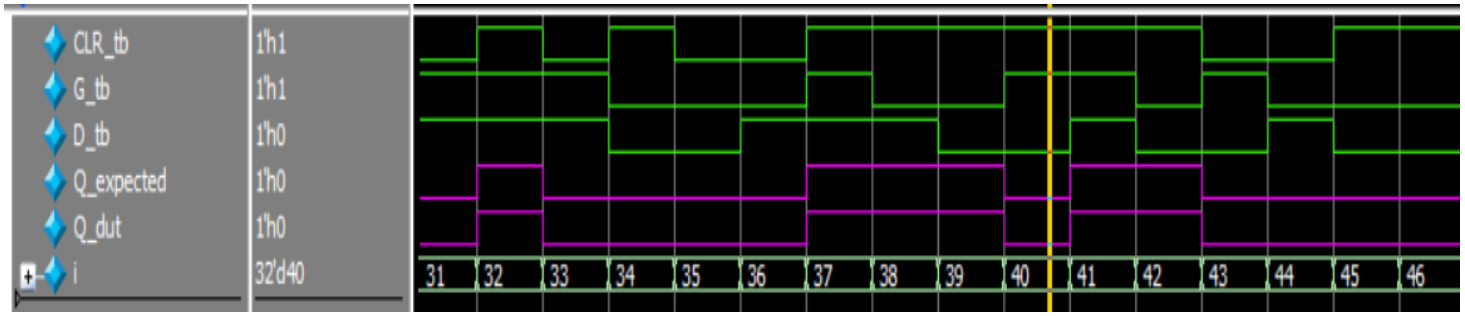
CLR	G	D	Q
0	X	X	0
1	0	X	Q
1	1	D	D

The testbench code:

```
module Data_Latch_tb();
reg CLR_tb,G_tb,D_tb,Q_expected;
wire Q_dut;
Data_Latch m0(CLR_tb,D_tb,G_tb,Q_dut);
integer i;
  initial begin
  CLR_tb=0;
  G_tb=0;
  D_tb=0;
  Q_expected=0;
  #10
  if(Q_expected!=Q_dut) begin
    $display("Errorrrr");
    $stop;
  end
  for(i=0;i<99;i=i+1) begin
    CLR_tb=$random;
    G_tb=$random;
    D_tb=$random;
```

```
      if (~CLR_tb)
        Q_expected=0;
      else if(G_tb)
        Q_expected=D_tb;
    #10
    if(Q_expected!=Q_dut) begin
      $display("Errorrrr");
      $stop;
    end
   end
   $stop;
   end
endmodule
```





[Q2]

A. Implement T-type (toggle) Flipflop with active low asynchronous reset. T-Flipflop has input t, when t input is high the outputs toggle else the output values do not change.
   - Inputs: t, rstn, clk
   - Outputs: q, qbar

(A)

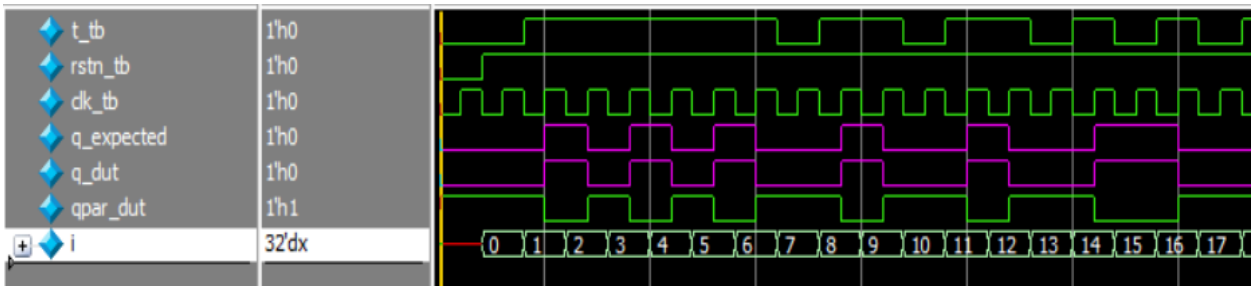
The design code:

```
module T_Flipflop(t,rstn,clk,q,qpar);
input t,rstn,clk;
output reg q;
output qpar;
assign qpar = ~q ;
always @(posedge clk or negedge rstn) begin
        if (~rstn)
          q<=0;
        else if (t)
          q<=~q;
end
endmodule
```

The testbench code:

```
module T_Flipflop_tb();
reg t_tb,rstn_tb,clk_tb,q_expected;
wire q_dut,qpar_dut;
T_Flipflop dut(t_tb,rstn_tb,clk_tb,q_dut,qpar_dut);
  initial begin
    clk_tb=0;
    forever
    #1 clk_tb=~clk_tb;
  end
  integer i;
  initial begin
    rstn_tb=0;
    t_tb=0;
    q_expected=0;
    @(negedge clk_tb);
    if(q_expected!=q_dut) $stop;

    rstn_tb=1;
```

```
      for(i=0;i<1000;i=i+1)begin
        t_tb=$random;
        @(posedge clk_tb);
        if(t_tb)   q_expected=~q_expected;
        @(negedge clk_tb);
        if (q_expected!=q_dut) $stop;
        end
$stop;
end
endmodule
```



B. Implement Asynchronous D Flip-Flop with Active low reset

- Inputs: d, rstn, clk
- Outputs: q, qbar

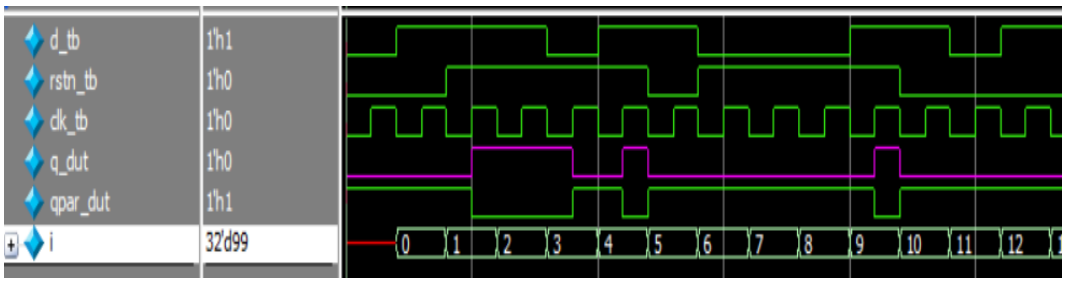
(B)

The design code:

```
module D_Flipflop(d,rstn,clk,q,qpar);
input d,rstn,clk;
output reg q;
output qpar;
assign qpar = ~q ;
  always @(posedge clk or negedge rstn) begin
          if (~rstn)
            q<=0;
          else
            q<=d;
  end
endmodule
```

The testbench code:

```
module D_Flipflop_tb();
reg d_tb,rstn_tb,clk_tb;
wire q_dut,qpar_dut;
D_Flipflop dut(d_tb,rstn_tb,clk_tb,q_dut,qpar_dut);
  initial begin
  clk_tb=0;
  forever
  #1 clk_tb=~clk_tb;
  end
integer i;
  initial begin
    rstn_tb=0;
    d_tb=0;
    @(negedge clk_tb);
    for(i=0;i<99;i=i+1) begin
      rstn_tb=$random;
      d_tb=$random;
      @(negedge clk_tb);
    end
  $stop;
  end
endmodule
```



(C)

C. Implement a parameterized asynchronous FlipFlop with Active low reset with the following specifications.

- Inputs: d, rstn , clk
- Outputs: q, qbar
- Parameter: FF_TYPE that can take two valid values, DFF or TFF. Default value = "DFF". Design should act as DFF if FF_TYPE = "DFF" and act as TFF if FF_TYPE = "TFF". When FF_TYPE equals "DFF", d input acts as the data input "d", and when FF_TYPE equals "TFF", d input acts the toggle input "t".

The design code:

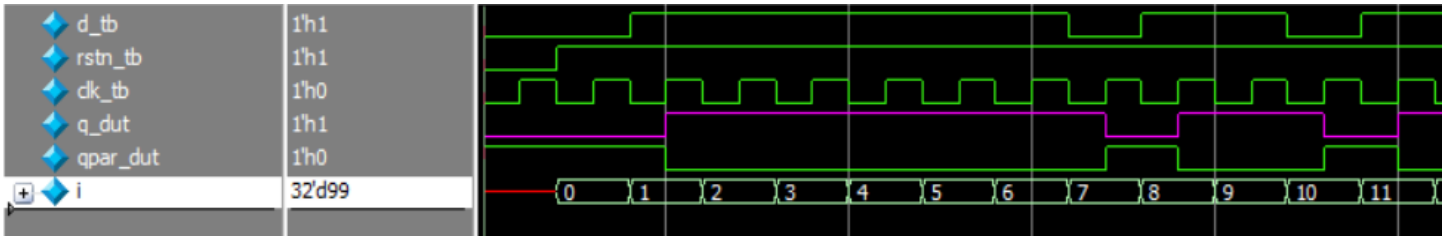
```
module Flipflop(d,rstn,clk,q,qpar);
parameter FF_TYPE="DFF";
input d,rstn,clk;
output reg q;
output qpar;
assign qpar = ~q ;
  always @(posedge clk or negedge rstn) begin
          if (~rstn)
            q<=0;
          else if (FF_TYPE=="TFF")
            q<=q^d; //when d is high then q is toggle
    else
      q<=d;
  end
endmodule
```

The testbench code:

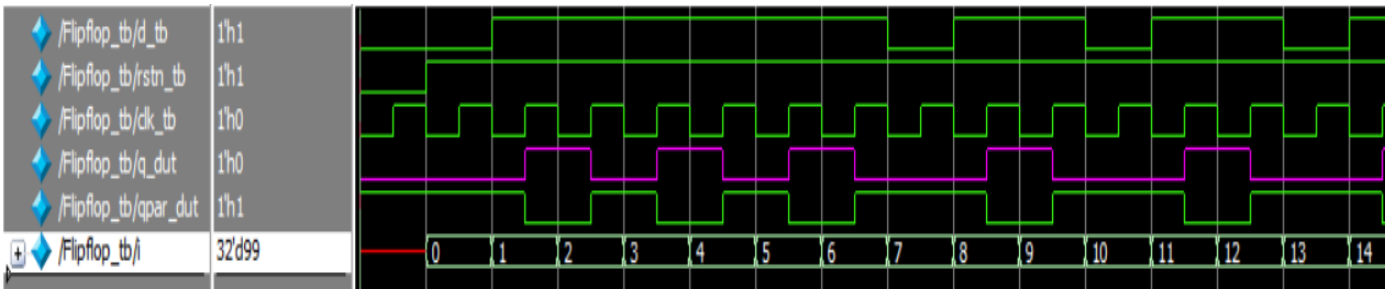
```
module Flipflop_tb();
parameter FF_TYPE_tb="DFF";
reg d_tb,rstn_tb,clk_tb;
wire q_dut,qpar_dut;
Flipflop #(FF_TYPE_tb) dut(d_tb,rstn_tb,clk_tb,q_dut,qpar_dut);
  initial begin
    clk_tb=0;
    forever
    #1 clk_tb=~clk_tb;
  end
integer i;
  initial begin
    rstn_tb=0;
    d_tb=0;
    @(negedge clk_tb);
    rstn_tb=1;
    for(i=0;i<99;i=i+1) begin
      d_tb=$random;
      @(negedge clk_tb);
```

```
    end
$stop;
end
endmodule
```

Wave of DFF:



Wave of TFF:



D. Test the above parameterized Design using 2 testbenches, testbench 1 that overrides the design with FF_TYPE = “DFF” and the testbench 2 overrides parameter with FF_TYPE = “TFF”

- Testbench 1 should instantiate the design of part B. as a golden model to check for the output of the parameterized design with FF_TYPE = “DFF”
- Testbench 2 should instantiate the design of part A. as a golden model to check for the output of the parameterized design with FF_TYPE = “TFF”

(D)

The code of DFF:

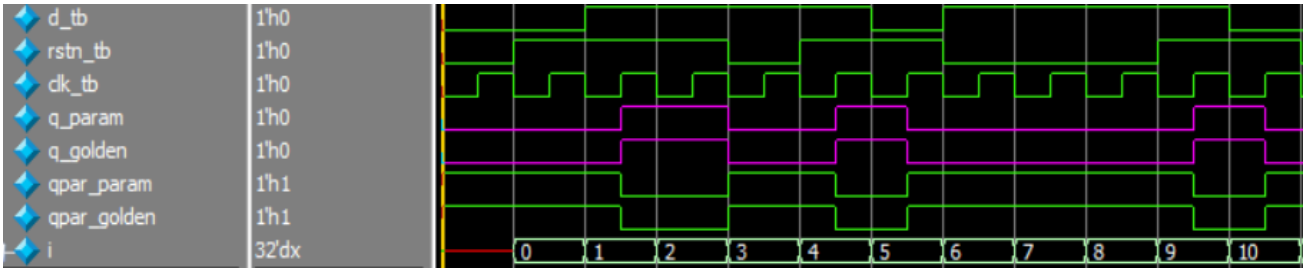
```
module Qd_for_DFF();
parameter FF_TYPE_tb="DFF";
reg d_tb,rstn_tb,clk_tb;
wire q_param,q_golden,qpar_param,qpar_golden;
D_Flipflop golden(d_tb,rstn_tb,clk_tb,q_golden,qpar_golden);
Flipflop #(FF_TYPE_tb) param(d_tb,rstn_tb,clk_tb,q_param,qpar_param);
  initial begin
    clk_tb=0;
    forever
    #1 clk_tb=~clk_tb;
  end
integer i;
  initial begin
    rstn_tb=0;
    d_tb=0;
    @(negedge clk_tb);
    if((q_golden!=q_param)||(qpar_golden!=qpar_param))  $stop;
    for (i=0;i<1000;i=i+1) begin
      d_tb=$random;
      rstn_tb=$random;
      @(negedge clk_tb);
    if((q_golden!=q_param)||(qpar_golden!=qpar_param))  $stop;
    end
    $stop;
  end
endmodule
```

The code of TFF:

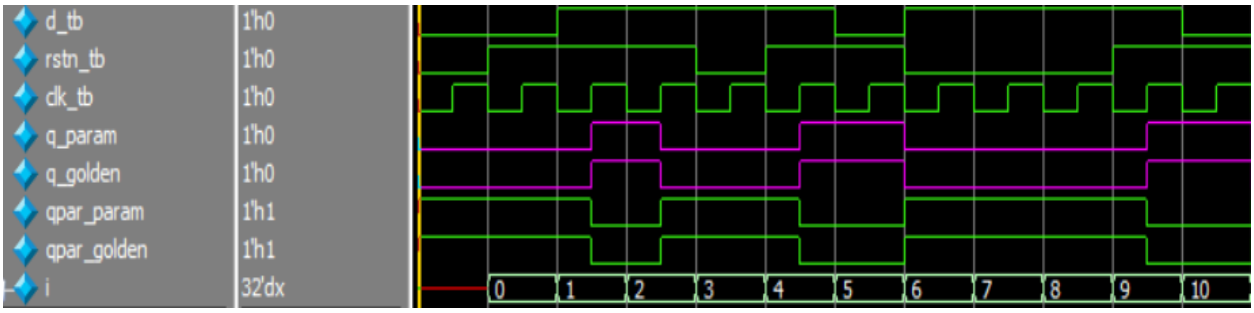
```
module Qd_for_TFF();
parameter FF_TYPE_tb="TFF";
reg d_tb,rstn_tb,clk_tb;
wire q_param,q_golden,qpar_param,qpar_golden;
T_Flipflop golden(d_tb,rstn_tb,clk_tb,q_golden,qpar_golden);
Flipflop #(FF_TYPE_tb) param(d_tb,rstn_tb,clk_tb,q_param,qpar_param);
  initial begin
    clk_tb=0;
    forever
```

```
    #1 clk_tb=~clk_tb;
  end
integer i;
  initial begin
    rstn_tb=0;
    d_tb=0;
    @(negedge clk_tb);
    if((q_golden!=q_param)||(qpar_golden!=qpar_param))  $stop;
    for (i=0;i<99;i=i+1) begin
      d_tb=$random;
      rstn_tb=$random;
      @(negedge clk_tb);
    if((q_golden!=q_param)||(qpar_golden!=qpar_param))  $stop;
    end
    $stop;
  end
endmodule
```

Wave of DFF:

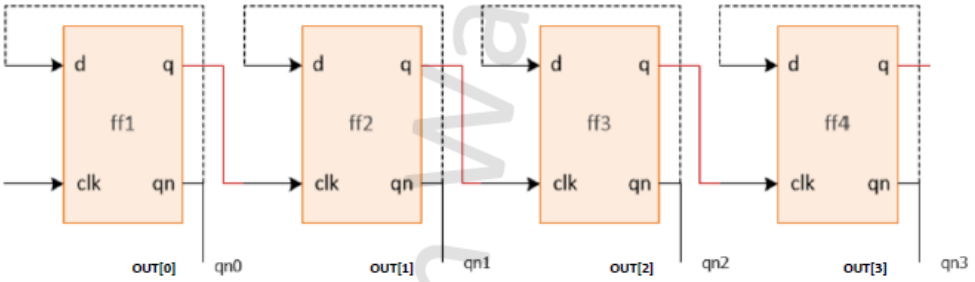


Wave of TFF:



[Q3]

3) Implement the 4-bit Ripple counter shown below using structural modelling (Instantiate the Dff from question 2 part B where the output is taken from the qn as shown below)
   - Inputs: clk, rstn;
   - Outputs: [3:0] out;



The design code:

```
module Ripple_counter(clk,rstn,out);
input clk,rstn;
output [3:0] out;
wire q0,qn0,q1,qn1,q2,qn,q3,qn3;
D_Flipflop DFF0(qn0,rstn,clk,q0,qn0);
D_Flipflop DFF1(qn1,rstn,q0,q1,qn1);
D_Flipflop DFF2(qn2,rstn,q1,q2,qn2);
D_Flipflop DFF3(qn3,rstn,q2,q3,qn3);
assign out = {qn3,qn2,qn1,qn0};
endmodule
```

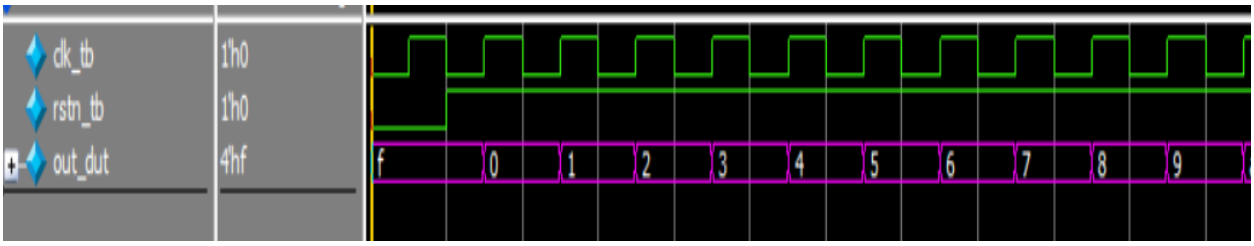
The testbench code:

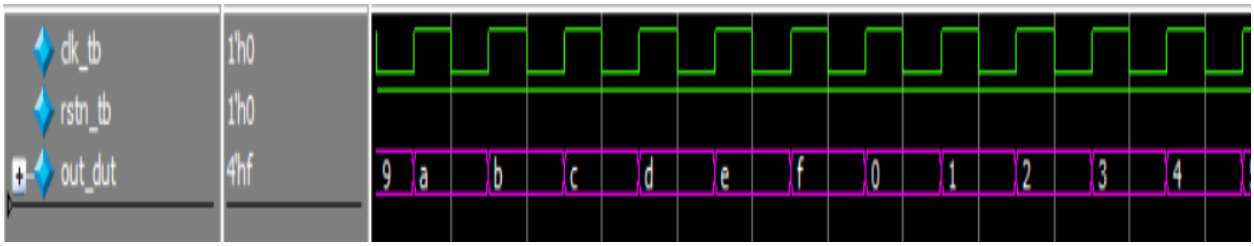
```
module Ripple_counter_tb();
reg clk_tb,rstn_tb;
wire [3:0] out_dut;
Ripple_counter dut(clk_tb,rstn_tb,out_dut);
  initial begin
    clk_tb=0;
     forever
     #1 clk_tb=~clk_tb;
  end
  initial begin
     rstn_tb=0;
     @(negedge clk_tb);
     rstn_tb=1;
     #200
     $stop;
  end
endmodule
```

The do file code:

```
vlib work

vlog D_Flipflop.v Ripple_counter.v Ripple_counter_tb.v

vsim -voptargs=+acc work.Ripple_counter_tb

add wave *

run -all

#quit -sim
```

```
QuestaSim> do run_ripple.do
# ** Warning: (vlib-34) Library already exists at "work".
# Errors: 0, Warnings: 1
# QuestaSim-64 vlog 2021.1 Compiler 2021.01 Jan 19 2021
```





[Q4]

4) Implement the following Parameterized Shift register

- Parameters

Name	Value	Description
LOAD_AVALUE	Integer > 0	Value loaded with aset is high
SHIFT_DIRECTION	"LEFT" or "RIGHT"	Direction of the shift register. Default = "LEFT"
LOAD_SVALUE	Integer > 0	Value loaded with sset is high with the rising clock edge
SHIFT_WIDTH	Integer > 0	Width of data[] and q[] ports

Default value for LOAD_AVALUE and LOAD_SVALUE is 1. SHIFT_WIDTH default value is 8.

- Ports

Name	Type	Description
sclr	Input	Synchronous clear input. If both sclr and sset are asserted, sclr is dominant.
sset		Synchronous set input that sets q[] output with the value specified by LOAD_SVALUE. If both sclr and sset are asserted, sclr is dominant.
shiftin		Serial shift data input
load		Synchronous parallel load. High: Load operation with data[], Low: Shift operation
data[]		Data input to the shift register. This port is SHIFT_WIDTH wide
clock		Clock Input
enable		Clock enable input
aclr		Asynchronous clear input. If both aclr and aset are asserted, aclr is dominant.
aset		Asynchronous set input that sets q[] output with the value specified by LOAD_AVALUE. If both aclr and aset are asserted, aclr is dominant.
shiftout	Output	Serial Shift data output
q[]		Data output from the shift register. This port is SHIFT_WIDTH wide

Notes:

1- Enable signal is dominant over the synchronous control signals "sclr and sset". However, the synchronous control signals "sclr and sset" are dominant over the load signal.
2- shiftout output represents the bit removed of the register and not the most significant bit.

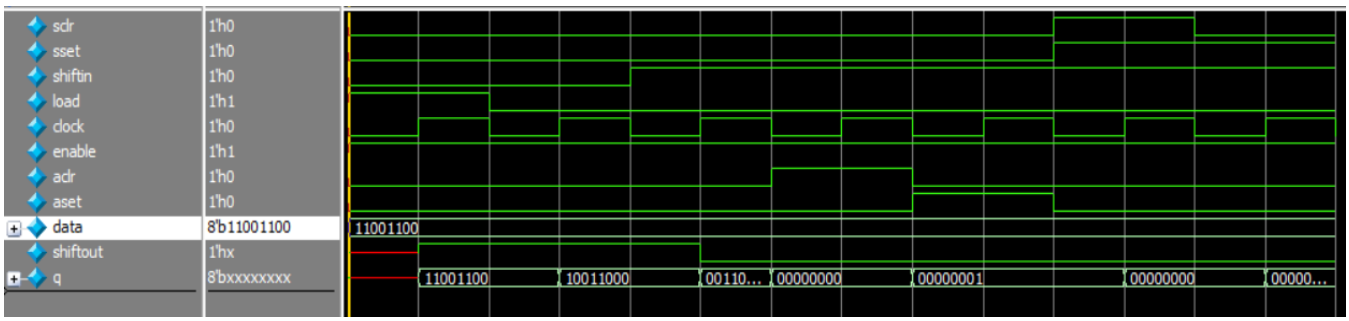
The design code:

```
module Param_Shift_register(sclr,sset,shiftin,load,data,clock,enable,aclr,aset,shiftout,q);
 parameter SHIFT_WIDTH=8,LOAD_AVALUE
=1,LOAD_SVALUE=1,SHIFT_DIRECTION="LEFT";
 input sclr,sset,shiftin,load,clock,enable,aclr,aset ;
 input [SHIFT_WIDTH-1:0] data;
 output shiftout;
 output  [SHIFT_WIDTH-1:0] q;
 reg temp_shiftout;
 reg [SHIFT_WIDTH-1:0] temp_q;
   always @(posedge clock or posedge aclr or posedge aset) begin
            if(aclr)
        temp_q <= 0;
      else if (aset)
        temp_q <=LOAD_AVALUE;
       else if (enable) begin
                  if(sclr)
             temp_q<=0;
          else if(sset)
             temp_q<=LOAD_SVALUE;
          else if(load)
             temp_q<=data;
          else begin
             if(SHIFT_DIRECTION=="Right")
                  {temp_q,temp_shiftout}<={shiftin,temp_q};
             else
                  {temp_shiftout,temp_q}<={temp_q,shiftin};
          end
       end
```

```
   end
 assign q = temp_q ;
 assign shiftout = temp_shiftout ;
endmodule
```

The testbench code:

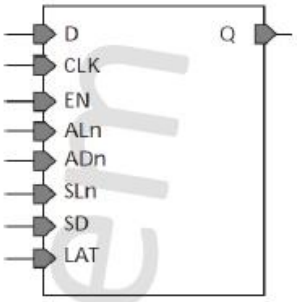
```
module Param_Shif_tb;
 parameter SHIFT_WIDTH=8,LOAD_AVALUE
=1,LOAD_SVALUE=1,SHIFT_DIRECTION="LEFT";
 reg sclr,sset,shiftin,load,clock,enable,aclr,aset;
 reg [SHIFT_WIDTH-1:0] data;
 wire shiftout;
 wire [SHIFT_WIDTH-1:0] q;
 Param_Shift_register #(SHIFT_WIDTH,LOAD_AVALUE,LOAD_SVALUE,SHIFT_DIRECTION)
dut(sclr,sset,shiftin,load,data,clock,enable,aclr,aset,shiftout,q);
   initial begin
      clock=0;
      forever
      #1 clock=~clock;
   end
   initial begin
      sclr=0;sset=0;shiftin=0;load=1;data=8'b11001100;//load the nubmber to register
      enable=1;aclr=0;aset=0; @(negedge clock);
      load=0; @(negedge clock);//shift left with shiftin=0
      shiftin =1; @(negedge clock);//shift left with shiftin=1
      aclr=1; @(negedge clock);// asynchronouns clear
      aclr=0;aset=1; @(negedge clock);// asynchronouns set
      aset=0;sclr=1;sset=1; @(negedge clock);// synchronous clear
      sclr=0; @(negedge clock);// anchronous set
      $stop;
   end
endmodule
```



[Q5]

5) Implement the following SLE (sequential logic element)

The design code:




```
module SLE(D,CLK,EN,ALn,ADn,SLn,SD,LAT,Q);
 input D,CLK,EN,ALn,ADn,SLn,SD,LAT;
 output  Q;
 reg Q_ff,Q_latch;
   always @(posedge CLK or negedge ALn ) begin
      if (~ALn)
         Q_ff<=~ADn;
      else if(EN&&~(LAT)) begin
         if (~SLn)
            Q_ff<=SD;
         else if(SLn)
            Q_ff<=D;
      end
   end
   always @(*) begin
      if(~ALn)
         Q_latch<=~ADn;
      if(LAT&&CLK&&EN) begin
         if (~SLn)
            Q_latch<=SD;
         else if(SLn)
            Q_latch<=D;
      end
   end
 assign Q = (LAT)?Q_latch:Q_ff;
endmodule
```

Input		Output
Name	Function	
D	Data	
CLK	Clock	
EN	Enable	
ALn	Asynchronous Load (Active Low)	Q
ADn*	Asynchronous Data (Active Low)	
SLn	Synchronous Load (Active Low)	
SD*	Synchronous Data	
LAT*	Latch Enable	

*Note: ADn, SD and LAT are static signals defined at design time and need to be tied to 0 or 1.

Truth Table

ALn	ADn	LAT	CLK	EN	SLn	SD	D	Q_{n+1}
0	ADn	X	X	X	X	X	X	!ADn
1	X	0	Not rising	X	X	X	X	Qn
1	X	0	↑	0	X	X	X	Qn
1	X	0	↑	1	0	SD	X	SD
1	X	0	↑	1	1	X	D	D
1	X	1	0	X	X	X	X	Qn
1	X	1	1	0	X	X	X	Qn
1	X	1	1	1	0	SD	X	SD
1	X	1	1	1	1	X	D	D

The testbench code:

```
module SLE_tb();
 reg D,CLK,EN,ALn,ADn,SLn,SD,LAT;
 wire    Q_dut;
 SLE   dut(D,CLK,EN,ALn,ADn,SLn,SD,LAT,Q_dut);
   initial begin
      CLK=0;
      forever
```

```
    #1 CLK=~CLK;
    end
 integer i;
  initial begin
    SD=1;ADn=0;LAT=0;ALn=0;//note SD&ADn&LAT are static
    D=0;EN=0;SLn=0;
    @(negedge CLK);
    ALn=1;
    for(i=0;i<100;i=i+1)begin
      D=$random;
      EN=$random;
      SLn=$random;
      @(negedge CLK);
    end
    ALn=0;D=0;EN=0;SLn=0;// I will reset the values to change the LAT from 0to1
    @(negedge CLK);
    ALn=1;LAT=1;
    for(i=0;i<100;i=i+1)begin
      D=$random;
      EN=$random;
      SLn=$random;
      @(negedge CLK);
    end
  $stop;
  end
endmodule
```